1 Standards (STD)

There are two standards prepared in support of the MIS/DSS MEDSTAT production environment. Both standards have been reviewed by HHSDC and ITSD for conformance to existing standards within those environments. A brief overview of each is presented below.

Process #	Process Name	Brief Overview
STD 1	JCL Standards	The purpose of standards for Job Control Language (JCL) is to promote standardization when accessing the mainframe environment and to conform to existing standards in place at the HHSDC and ITSD.
STD 2	COBOL Standards	The purpose of COBOL coding standards is to promote consistency, readability and ease of maintenance of all custom programs designed for the MIS/DSS. Having this standard also helps to ensure that COBOL programs have been written in conformance to existing standards in place at the HHSDC and ITSD.

Table of Contents

1. Job Control Language (JCL) Standards

1.1 Overview

The purpose of the JCL Standards document is to promote standardization when accessing the mainframe environment. This includes:

- Confirming that JCL promoted to the Test and Production regions meets environment requirements
- Verifying that technical resource utilization is optimized
- Assuring that appropriate updates are made to the operations scheduling mechanism to minimize resource contention
- Achieving maximum operational design which minimizes elapsed time of batch jobs and systems
- Assuring that any deviation from the operational standards and guidelines presented herein are documented

MEDSTAT is obligated to plan for, install, and run operational systems that meet the demanding service levels of the Management Information System/Decision Support System (MIS/DSS) Project ("the Project"). These systems will run on hardware installed specifically for this purpose at the Health and Human Services Data Center (HHSDC). This particular document states the recommended methods and guidelines for project JCL, incorporating HHSDC standards, as applicable***. The use of standardized job and procedure names, however, will only apply to the custom JCL specifically developed for Medi-Cal — all other standards described in this document will apply to both custom and core JCL used for the Medi-Cal MIS/DSS project.*** The document has been compiled to facilitate communication between project database, operations, and development staff members, thus improving overall quality of the operational environment. It will also be used to facilitate communication and transitions of responsibility between MEDSTAT, HHSDC, DHS and Information Technology Services Division (ITSD) staff members.

Appropriate application of these guidelines will enable us to:

- Ensure effective and efficient utilization of technical resources
- Minimize potential operational errors
- Reduce migration and operations preparation time frames
- Promote assimilation of new personnel

1.2 Purpose

This document serves as a central document that can be accessed by all developers to communicate and enforce common methods and styles of JCL development. This process is intended to mitigate production errors by utilizing consistent practices in the development of jobs and procedures.

1.3 Scope

JCL is used to invoke, control, and on occasion, terminate the execution of system and application programs. These guidelines are focused specifically on JCL, JCL-related objects, and other related operational components targeted for use in the IBM mainframe environment. Although the document is intended for use by all Developers and Database/Operations representatives, it does not address specific development and unit test operational issues, such as disposition of testing data sets.

The document establishes baseline rules for all components and parameters within the following:

- JOB statement
- PROC statement
- EXEC statement
- DD statement
- Data Set Attributes

Additional components and considerations included in this documentation are listed below.

- Back-up
- Performance
- Job design
- Documentation
- Software standards

1.4 Responsibility and Enforcement

The Development Team Lead is responsible for ensuring that all new jobs and existing job modifications conform to these guidelines.

1.5 General Considerations

There are no general considerations for this process.

1.6 Skill Requirements

The skills required to utilize this process include a mastery of the JCL language sufficient enough to be allowed to implement new jobs/procedures into production with peer evaluation/review. This level of expertise would likely be obtained after 6 months to 1 year of mentored development in an IBM MVS mainframe environment.

1.7 Entry Criteria

This process is entered any time a new policy or process needs to be drafted or finalized.

1.8 Procedure Steps

The following steps will be followed when creating new and modifying existing JCL objects.

JCL Format

All JOB, PROC, EXEC, and DD statements will conform to the format shown in Table 1.8-1. Each parameter will be coded on a separate line[1].

Table 1.8-1. JCL Format

Field Name	Description/Contents	Column Position
Identifier Field	//	1-2
Name Field	**JOBname** **STEPname** **DDname**	3-10
Operations Field	**DD** **JOB** **EXEC**	12-15
Parameter Field	**on JOB and DD statements** **on EXEC statement**	Starts in Column 16 Starts in Column 16 (IR 1047)
Comment Field or end of JOB	**//***	1-3
End of Data	**/***	1-2

Table 1.8-1

1.9 Execution JCL

This section covers creation and maintenance of Execution JCL.

1.9.1 JOB Statement

This section describes the parameters and contents of the execution JCL JOB statement. It is recommended (not required) that all keyword parameters be positioned in alphabetic order. Figure 1.9.1-1 shows an example of a correctly coded JOB statement. ***NOTE***: ***Per the 11/22/99 E-mail from Don Knifong (ITSD) MEDSTAT should modify the accounting information in***

[1] The requirement for parameters on separate lines applies specifically to JCL to be delivered to HHSDC and is not enforced at the project level.

all production jobs (HMHCP7300P) and development (HMHCP7200T) in order to provide a means of identification of MEDSTAT submitted jobs if this system is ever integrated into an environment shared by other state resources.

For example:

```
COLS+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
//JEDIT        JOB    (Job Accounting = HMHCP7300P),
                    'name',
                    CLASS=A,
                    MSGCLASS=J,
                    USER=XYZPROG
```

Figure 1.9.1-1
JOB Statement Example

1.9.2 JOB Name

Job names reflect an easily recognized pattern to allow easy tracking and status display. To promote standardization of production job names, the following conventions will be utilized.

JOB name ="HMxyznnn"

where:
H = HHSDC (constant)
M = MEDSTAT (constant)
x = Product/Application
y = Data Source
z = Function/Process
nnn = Job number frequency (unique job ID)

EXAMPLE: HMDC310 (see detailed legends below)

H = JOB created for execution at HHSDC
M = MEDSTAT
D = DataScan
C = Claim Data Source
3 = Edit Function
10 = First on-request job in the function series

Product/Applications

D = DataScan
M = Performance Measurement Workstation (PMW)
N = Non-Product (MEDSTAT) Specific Application
P = Panorama

U = Utility or Third Party Software

Data Source(s)

A = All Claims (Drug and Medical)
C = Medical Claims (MCLM)
D = Drug Claims (DCLM)
E = Eligibility
F = Managed Care Financials
M = Managed Care Capitation
N = Managed Care Provider
V = Provider (PMF/PLF)

Functions/Processes

0 = Pre-Convert
1 = Convert
2 = Install
3 = Edit
4 = Build
5 = Update
6 = Rolloff/Delete

Frequency	JOB Number Series
Daily	000-099
Semi- or Tri-weekly	100-149
Weekly	150-249
Semi-Monthly	250-299
Monthly	300-499
Quarterly	500-549
Semi-Annual	550-599
Annual	600-699
Intermittent/On Request	700-999

Job Numbers should start with '010' and be incremented initially by 10 (010, 020, etc.) to allow new jobs to be inserted as required without forcing renumbering of existing jobs. If it is anticipated that the system or application in a particular series will have a large number of unique jobs, the initial numbering increment will be 005 (005, 010, 015, etc.) Any JOBs that cannot conform to this criteria will use the number series which most closely matches its required frequency.

1.9.3 JOB Class

JOB class is a one-character alphanumeric which identifies the job service requirements. Specific JOB classes are defined to control the assignment of system resources to each job. JOB classes are automatically assigned by the combination of required CPU time and number of required tape drives.

The following table (Table 1.9.3-1) identifies HHSDC approved job classes.

CLASS	REQUIRED RESOURCES		
	CPU TIME (MINUTES)	# TAPE DRIVES	QUALIFIERS
A			Not for MEDSTAT Usage
B	No Limit		Not for MEDSTAT Usage
C			Not for MEDSTAT Usage
D	No Limit		Not for MEDSTAT Usage
E			HHSDC Use Only[2]
F	No Limit		HHSDC Use Only
G	No Limit	No Limit	JOB submitted by ESP[3]
H	No Limit	No Limit	JOB submitted by ESP and has a /*WHILE Card
K	N/A	N/A	HHSDC Use only (EDP007 to print spooled data)
M	No Limit		Has a /*WHILE Card[4]
P			Not for MEDSTAT Usage
Q			Not for MEDSTAT Usage
R			Has a /* WHILE Card
S	N/A	N/A	HHSDC Software
X	N/A	N/A	CICS Regions
Y	N/A	N/A	Production

Table 1.9.3-1
HHSDC Approved JOB Classes

1.9.4 Message Class

Message class is a one-character alphanumeric which identifies the job output repository and disposition. Figure 1.9.4-1 illustrates the coding of the message class. Refer to Table 5-5 , "Approved SYSOUT Classes" for approved message class values.

```
COLS+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
//HMJEDIT       JOB    (Job Accounting),
                'name',
                CLASS=1,
                MSGCLASS=J,
                USER=XYZPROG
```

Figure 1.9.4-1
Message Class Coding Example

[2] For all items marked "HHSDC Use Only" - At time of publication, these options are not supported on MEDSTAT-S4
[3] ESP is the current automated JOB scheduling system
[4] Confirm with Database Operations prior to using this option

1.9.5 USER Parameter

The USER parameter is hard-coded on the JOB statement and must be included for all scheduled jobs. The USER ID controls the security levels for the JOB, and therefore must be defined to the system security package. Project USER IDs adhere to the following format:

- Position 1-2 must contain an "HM" (MEDSTAT)
- Position 3-7 must contain two to five user-defined characters

USER IDs are assigned by the Database/Operations group.

1.10 JCL Procedures (PROCs)

1.10.1 Design Guidelines

It is recommended that catalogued procedures be used rather than running in-stream submission JCL or dynamically created JCL. Acceptable exceptions are jobs running in the development environment while JOB/PROC design are still under operational analysis.

Embedded procedures (PROCs within PROCs) are to be avoided. If it is necessary that they be used, they will be limited to one level deep, and restricted to processes that span applications, such as SAS, SORT, and other utilities.

Production processes are limited to the execution of a single application process or function with common utilities within an execution JCL. This will facilitate prompt operational problem identification and resolution.

All production application processes must exist in the form of catalogued procedures. PROCs must be designed so that all non-critical processes are executed outside of the critical path schedule. For example, if a step within the procedure is itself not critical, and has no immediate successive dependencies, that step must be incorporated into a separate PROC.

1.10.2 JCL Procedure (PROC) Names

All production PROCs will reside in the appropriate procedure library (PROCLIB). These libraries are partitioned data sets (PDS). Each PROC is a member in the PDS. The PROC member names will be created by the Developer.

1.10.3 PROC Description

It is recommended that a PROC description be initially developed as header/comment information within each procedure. When the procedure is moved into a test or production

environment, the description will communicate to the Database/Operations staff the requirements and use of the PROC.

1.10.4 EXEC Statement

The following rules apply when coding the EXEC statement:

- JOB step names must begin with "JS" followed by an ascending three digit number indicating the sequential number of the step.

- PROC step names must begin with "PS" followed by an ascending three digit number indicating the sequential number of the step.

NOTE: When numbering steps, leave gaps within the step numbers to allow later insertion of steps not initially anticipated. The size of the gaps must correspond to the number of steps in the job.

If a PARM field is required, it must be coded as a continuation on a new line.

If a REGION parameter is required, it must always be coded on the EXEC statement and never on the JOB card. The REGION parameter must be allocated as 0M. This allows the operating system to determine the necessary storage and dynamically allocate the proper amount. If a program requires more than 5120K, consideration must be given to redesigning the program to include the use of extended or expanded memory. Programs of this size may cause serious system performance degradation, or contention issues.

Condition Code Logic can be used on the EXEC statement. Use condition codes to control the conditional execution of subsequent steps.

1.10.4.1 DD Statement

DD statements must be grouped in the following categories in a catalogued procedure:

- Software dependent data sets (e.g., DB2, CICS libraries, etc.)
- Databases
- Input data sets
- Update "in place" data sets (Input/Output)
- Temporary work areas
- Output data sets
- SYSOUT data sets
- SYSUDUMP and SYSPRINT statements (compulsory), plus any other installation required print statements (such as SYSABEND).

If coded, output DD statements must adhere to the following component/parameter order:

- Line 1 - Data Set Name (DSN)
- Line 2 - Disposition (DISP)
- Line 3 – DCB parameters (DSORG, RECFM, LRECL, BLKSIZE)
- Line 4 - UNIT
- Line 5 - SPACE Allocation (SPACE)
- Line 6 - Tape Label Information (LABEL)
- Line 7 - Any remaining keywords

This practice will assist in expeditious JCL review, problem identification, and problem resolution.

1.10.5 Data Sets in JCL

1.10.5.1 Data Set Naming Conventions

Data set naming promotes simplification of operational issues. High level qualifiers promote rapid identification of the nature of the data set (development, test, production, etc.), as well as the security requirements. This will allow greater flexibility in the security coding rules, and simplify security maintenance, as well as support data set recovery rules. To quickly identify the origin of the data set, the data set name will have the JOB and Step number imbedded.
The approved Data Set Naming practice is:

HLQ.VERSION.RUNQUAL.XXXXXXXX.YYYYYYYY.ZZZZZZZZ

Where:

HLQ =	High Level Qualifier (See List)
VERSION =	The DataScan version for test jobs or “PMED” for production
RUNQUAL =	For installation mode (build jobs), the phase of the build (e.g., P31) For monthly update, mmmyypp (the month, year and phase of the Update eg. JUN98P3)
XXXXXXXX =	Data Descriptive or Functional Name (Level 1)
YYYYYYYY =	Data Descriptive or Functional Name (Level 2)
ZZZZZZZZ =	Data Descriptive or Functional Name (Level 3) as needed

EXAMPLE:

HM.TMED.V4R01.P33.SORTED.CLAIMS

HIGH LEVEL QUALIFIERS	DESCRIPTION
HM.TMED	Development Application Libraries
HM.PMED	System/Acceptance/Prod Application Libraries
HM.MIGR	Migration/Staging Libraries
HM	non-Development Data Sets
Userid	TSO Development User IDs and Development Data Sets
SYS1	System Software Libraries

Table 1.10.5-1
List of High Level Qualifiers

NOTE: VSAM component names will be the same as the internal cluster name with an additional Low Level Qualifier (LLQ).

For example:

Cluster Name: HLQ.VERSION.RUNQUAL.[CLUSTERNAME]
Data Component: HLQ.VERSION.RUNQUAL.[CLUSTERNAME].Data
Index Component: HLQ.VERSION.RUNQUAL.[CLUSTERNAME].Index
Alternate Index: HLQ.VERSION.RUNQUAL.[CLUSTERNAME].AIX
Path: HLQ.VERSION.RUNQUAL.[CLUSTERNAME].Path

NOTE: The use of aliases is highly discouraged; aliases are not recommended for project use.

1.10.6 DISP Parameter

The following rules apply when coding the DISP parameter:

- Catalogue all new data sets, including tape data sets, which are to be kept past the end of the creation job. Data sets not catalogued will be deleted.
- Code the following for data sets to be catalogued:
  DISP=(New,CATLG,DELETE)
- Do not code the following for catalogued data sets:
  DISP=(NEW,KEEP) or
  DISP=(NEW,KEEP,KEEP)
- Temporary data sets are utilized as needed, and exist only for the life of the creation step; all other temporary data sets must be allocated, and treated as permanent data

sets. These data sets must also be deleted when no longer needed for current processing.

1.10.7 Unit (Storage Device)

When creating an output data set, specific device addresses must not be named. This is to avoid specific device dependency.

DEVICE TYPE	*GENERIC UNIT NAME*	*USAGE*
Tape (Cartridge-3490)	***TAPE***	***All in-house tapes and external tapes unless otherwise requested***
DASD	***SYSDA***	***Direct Access Devices***

Table 1.10.7-1
Generic Device Names

All tape drives default to the highest of their available densities on output.

The "AFF" parameter will be used when one or more tape or cartridge data sets are being read or written consecutively to the same device. This will prevent more units being allocated than are required.

To ensure uniformity of coding in the case of multiple-file backups, the "UNIT=AFF" statement must refer to the last data set that was previously backed up.

The "DEFER" parameter must be coded in the following circumstances:

- On input data sets that are allocated but may not necessarily be opened.
- On output data sets so that a cartridge or tape is only mounted if the program opens the file.

It is recommended that three or fewer tape drives be concurrently allocated per job step. Allocation of four (or more) tape drives may result in tape allocation problems or job queuing problems.

1.10.8 Volume

To reduce unnecessary mount activity, the "RETAIN" parameter must be coded when passing a tape or cartridge data set from one job step to another.

Volume "refer backs" may only be used if they do not impact the restart opportunities of the job.

When backing up multiple data sets to cartridge, the "refer back" must always relate to the DDNAME of the previous data set being backed up. This prevents job failure if the data sets expand over more than one volume.

Volume serial numbers may only be coded in the JCL for input data sets which are not known to the system catalogue, unless an output data set is to be written to a "stranger tape" (a tape or cartridge that is unknown to the current Tape management system).

1.10.9 Space Management for Disk Data Sets

Figure 1.10.9-1 shows an example of how to code the SPACE parameter.

```
COLS+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8
//DDSPACE     DD  DSN=&&TEMP,
                DISP=(NEW,DELETE),
                DSORG=PS,
                RECFM=FB,LRECL=1049,
                BLKSIZE=0,
                UNIT=&Unit,
                SPACE=(200,(100,50),RLSE)
//*
```

Figure 1.10.9-1
SPACE Parameter Coding Example

The following guidelines are to be used when creating data sets on Direct Access Sequential Devices (DASD).

Physical Sequential Data Sets

Allocate space equivalent to the maximum anticipated size of the production file. If data set is to be updated in place, allow sufficient space for insertions.

Partition Organization

Allocate space requesting megabytes up to a maximum of 100 megabytes. Specify the secondary allocation as 50% of the primary allocation with a maximum of 30 megabytes. Specify Directory blocks in multiples of 40 to prevent out of directory errors.

NOTE: It is not recommended that Partition Organization Extended (POE) be used for production data sets.

Direct Access

These files must allocate all needed space upon initial creation, and cannot be larger than 2.3 gigabytes in size.

For VSAM KSDS/ESDS data sets, specify primary space of 1 to 300 Mb, and *no secondary allocation.*

Sort Work Allocations

Allow at least three times the file size for sort work space allocations, with a minimum of three sort work DDs. The size of the sort work space allocations, if too small, may cause detrimental performance of the sort. For very large files, always code the sorted data sets to be catalogued.

1.10.10 LABEL

All tape or cartridge data sets must be created with a standard label. If the data set is required beyond the default retention period (3 days), code Retention Period (RETPD) instead of Expiration Date (EXPDT). If "RETPD" is used, all files on the same tape volume(s) must be coded for the same retention period. "RETPD" is now to be coded as a stand-alone parameter and is no longer coded as part of the LABEL parameter.

NOTE: Developers are responsible for managing their own tape data sets. The maximum retention period for development data sets must not exceed 60 days without explicit communication with Database/Operations. If a retention period of more than 60 days is required, a memo must be written to Database/ Operations requesting additional time, and containing justification for the extension. Additional information must include the data set names, the amount of time requested, and the contact person who owns the data set. To facilitate the above guidelines, Database/Operations will create and circulate a weekly report from the tape management system to the development staff which indicates retention periods about to be exhausted.

1.11 Backup Considerations

1.11.1 Frequencies

All applications will be reviewed to determine the practical limits on frequency of use and recoverability and/or restartability, including the estimated resource usage and associated CPU cost balanced against the time/cost incurred during application recovery and/or restart. Backup frequencies must be based on these determinations. Other factors to consider are the backup method, the size of the file(s) involved, the lag time between processing steps, the overall operational schedule, and the availability requirement for the file(s) for inter-job dependency and contingency purposes.

1.11.2 Data Set and Data Base

All application sub-systems must contain adequate backups of data sets or data bases to enable system recovery, with consideration for cost to the project.

Data set backup steps must process outside of the critical production path. The exception to this rule is data sets required for system recovery.

GDG Considerations

The use of GDG files must not impact the re-startability of jobs. Build GDG indices using the following IDCAMS statement:

DEFINE GDG(NAME(dsn) SCRATCH NOEMPTY LIMIT(x))

Where x = maximum number of generation data sets (GDGs) that can be associated with the GDG.

1.12 General Considerations

The use of Generation Data Groups (GDGs) is strongly encouraged to assist in the effective and efficient use of technical resources (TAPE and DASD).

The Catalogue index entry for GDG backups must reflect the expected life of the data set for all storage media. If a data set is created on a daily basis and retained for 14 days, then the index must be built for 14 generations.

This standard may be applied up to 255 generations, as this is the current limitation. If more than 255 generations are required, code the Retention Period to ensure that the data sets are retained and available, even though not in the catalogue.

Single currency data sets must not be merged onto a consolidated backup; they must be backed up individually.

Backups must be taken of all data sets sent off-site.

1.12.1 Performance Considerations

Tape sort work files must not be used under any circumstances for this project.
Single tape data sets are only read once in a job stream. If multiple reads are required, the data set must be copied to disk to support multiple accesses, unless prohibited by resource (size) requirement.
In many cases, the file definition of a VSAM file is critical to performance, and it is therefore essential that the definition is thoroughly justified in terms of CISIZE, use of buffers, VSAM file type, etc. Consider the following points:

- Use VSAM keywords to attain optimum performance. This will vary from file to file, depending on where or how the data set is accessed (sequentially or randomly).
- When defining a VSAM file, always use the most accurate average record length.
- When reading a VSAM KSDS file, use the AMP parameter in the JCL “DD” statement.

1.12.2 Job Design Considerations

Items included in this section are to be considered when designing all test and production JCL.

1.12.2.1 Ability to Rerun JOBs

JCL procedures must be written to enable automatic restart from the point of failure.

Any housekeeping or cleanup procedure must be minimized, as by standard, any files to be catalogued during execution must have been deleted when no longer needed, and only catalogued if the data set is to be kept.

Use an industry standard utility such as IDCAMS or IEFBR14 to delete data sets.

Any program that updates a database must take checkpoints in order to commit and log updates. Any program that runs for longer than 30 minutes elapsed time must checkpoint.

The application and job streams must be designed not to include manual intervention during execution.

1.12.3 Abend Handling

Restore JOBs

Where appropriate, specific restore procedures must be created and documented for VSAM files. These procedures will be used when a batch process is not automatically re-runable. They will take corrective actions prior to the resubmission of the original job.

Where appropriate, and with consideration for cost to the production environment, complete VSAM file backups prior to application update.

Determine at PROC design time if it is necessary to create a special restore PROC will be developed for recovery of any complex process.

Design and code programs to always return a condition code of zero (0) if the program completes successfully. For example, data exceptions will be handled and reported programmatically and not by a non-zero condition code. With the exception of branching requirements, any non-zero condition code will denote process failure that must result in execution interruption. Non-zero codes may be required for branching purposes within a job stream. Use JCL branching logic (IF/THEN/ELSE/ENDIF) processing whenever possible. Refer to the JCL Reference Manual for correct format and usage of JCL branching.

1.13 Table 1.13-1 - HHSDC Approved SYSOUT Classes

SYSOUT classes are assigned by HHSDC. The following table illustrates current HHSDC values to use for SYSOUT and Message Classes.

SYSOUT CLASS	TYPE OF FORM	DEVICES/USAGE
A	Standard[5]	All Printers
B	Standard	Punch
C	Standard	
F	Special	
I	Standard	XEROX
J	FICHE	COMUNIT
K	PRINTING AND PUNCHING REQUIRED	PUNCH
O	STAPLED AND FOLDED	XEROX
P	Standard	(PREFERABLY)
X	HELD[6]	ALL PRINTERS
Y	HELD[7]	ALL PRINTERS
0(zero)	CONDITIONAL PURGE[8]	ALL PRINTERS
	SPECIAL	
	SPECIAL	PUNCH
	SPECIAL	
	SPECIAL	(PREFERABLY)
	STANDARD	

Table 1.13-1
Approved Sysout Classes

1.14 General Operational Considerations

All non-critical processes must be executed outside of the critical path schedule. The following considerations must be taken into account when structuring the schedule.

Recoverability
Where possible, write recovery procedures to easily recreate files to be sent off-site or to external shops.
Two backup copies are to be taken of files that are required to recover an application system, one of which must be sent off-site.

[5] Standard printer form is defined as one part (14 7/8" wide by 11" long). Standard setup is ½" top and left margins. Standard printer form for the 3820 is for HHSDC use only.
[6] Output defined as CLASS X is automatically held for review through TSO. All SYSOUTs remaining in hold status after 48 hours are released at 2300 hours that night.
[7] Output defined as Class Y is automatically held for review through TSO. All SYSOUTs remaining in hold for two (2) weeks will be deleted at 2300 that night, and will never be printed.
[8] All SYSOUT '0'(Zero) classes will be purged after 6 days. Until then, it will be held in the output queue.

Disaster recovery and security requirements must adhere to audit requirements.

Tape or Cartridge Handling
All tape or cartridge data sets must be under the control of the Tape management system package.

1.15 Process Exceptions

Any exceptions to the recommendations in this guideline must be documented in the appropriate JCL object (JOB, PROC), and approved by the Development Manager.

1.16 Exit Criteria

Agreement by all members of the Core Team effected by the policy/process is required before the policy/process is implemented. Each Core Team member must be prepared to sign-off on the policy/process and inform their team members that they will support the implementation and enforcement of the content.

1.16.1 Exit Exception Criteria
By agreement of the Project Director, the format and content of this policy/process may be deviated from standard.

1.16.2 Exit Exception Handling
The exception must be documented and agreed to by the Project Director.

1.17 Reference material

N/A

1.18 Policy History

Established/Revision Date	Established/Revised By	IR No.	Change Description
11/30/99	JTM	1233	Update section 5.1 to indicate that JCL job naming conventions will apply only to custom JCL developed specifically for Medi-Cal and *not* the *Core* MEDSTAT DataScan, PV, nor PMW jobs. This change is made to eliminate potential confusion and unwarranted JCL updates that would occur as a result of MEDSTAT product upgrades

			(MEDSTAT Ann Arbor does not use Medi-Cal custom JCL naming standards). In addition, account code information will be standardized per HHSDC direction.
10/15/98	JTM	1047	Alter job number schema, start position in EXEC statement
8/18/98	Joe D	1047	Job naming convention, allow 0M for region, etc.

Table of Contents

1. COBOL Coding Standards

1.1 Overview

The purpose of the COBOL Coding Standards section is to promote consistency and reliability in the development and maintenance of COBOL programs by collecting and publishing all rules in a single location.

1.2 Purpose

The purpose of the COBOL Coding Standards is to promote consistency, readability, extendibility, and ease of maintenance to all custom programs designed for the Management Information System/Decision Support System (MIS/DSS) Project ("the Project").

Appropriate application of these Standards will assist in:

- Minimizing potential coding errors
- Ensuring effective and efficient utilization of technical resources
- Increasing individual and Developer team productivity
- Promoting assimilation of new personnel

These standards reflect a compilation of programming standards from The MEDSTAT Group, Department of Health Services (DHS), and the Information Technology Services Division (ITSD). The programs subjected to these standards for compliance are targeted for delivery to the Department, and may (eventually) be maintained by the Department staff.

1.3 Scope

This document will be used by any project team member responsible for developing COBOL programs used on the Project.

This standard addresses the procedural design of programs as well as actual COBOL Coding requirements.

1.4 Responsibility and Enforcement

Responsibility for reviewing and updating the COBOL Coding Standard and enforcement of the Standard across all Developers is the responsibility of the Development Manager. The actual evaluation of compliance of a given program may be delegated.

Developers are responsible for creating new and modifying existing code for COBOL conversion and custom programs. The Developers provide, where required, customized programs for specialized jobs to be executed in the test and production environments. They are also

responsible for promoting programs and other program-related components from the development libraries to the migration libraries.

The Database/Operations staff is responsible for ensuring that the appropriate promotions have occurred from the migration library to the test library, and from test to production.

Programs will be reviewed for compliance to this guideline at the time of the code walk-through. Programs not in compliance and where compliance exceptions have not been satisfactorily documented must be modified and re-submitted.

1.5 General Considerations

The strategy presented here intends to lead to a design model for conversion programming. Our strategy will promote quality and consistency throughout the programming process. To accomplish this, we encourage the use of structured programming techniques, and standardization of programming practices.

Structured programming is defined as the practice of constructing software according to a procedural architecture using a series of logical constructs. A program coded according to structured programming techniques usually incorporates the following elements:

- Top down design (hierarchical architecture)
- Controlled architecture ("Main-line" driver)
- Use of functional modules (functional paragraphs or sub-programs)
- Single entry and single exit points
- Common, re-useable routines
- Use of logical constructs (hierarchy, sequence, alternation, and repetition)
- Limited use of unconditional program transfers (e.g., no GO TO statements)

Use of structured programming promotes ease of maintainability and extendability to all programs. The structured constructs also limit the procedural design of programs and modules to a small number of predictable operations, which aid in defining and executing unit and string tests.

The use of a limited number of logical constructs also contributes to a human understanding process. The logical "chunks" allow a reader to recognize procedural elements of a module rather than reading the code line by line. Understanding is enhanced when readily recognizable logical forms are encountered, thereby increasing the self-documenting properties of the code.

This blueprint for software construction forms the basis for all conversion and custom developed code.

1.6 Skill Requirements

Developers creating code for use on this project should be familiar with the concepts of structured program development.

1.7 Entry Criteria

This process is entered any time a project COBOL component is created or altered.

1.8 Procedure Steps

The following steps outline the development process request and authorization to program.

- Investigation Request (IR) is generated
- Change Control process reviews for priority and scheduling
- Assignment(s) made to Data Manager and Developer
- Data Manager and/or Developer perform analysis
- Data Manager develops design specification
- Data Manager and Developer participate in design walk-through to ensure consistent understanding of requirements
- Developer makes changes, as needed
- Developer performs unit test
- Developer and Data Manager participate in code walk-through
- Developer reviews unit test with Data Manager/Lead Developer
- Lead Developer authorizes migration
- Developer migrates affected programs and related objects

The Developers will utilize the guidelines presented in this document whenever they generate and/or modify COBOL program code.

1.8.1 COBOL Coding Standards and Procedures

General Coding Guidelines

- Use EJECTs, SKIPs, and blank lines to improve program readability.

- Use indentation and align vertical columns to enhance readability and show relationships.

1.8.1.1 Identification Division

The first division of all COBOL programs is used to identify the program and to provide a high level overview of the program's history.

PROGRAM NAME
The format of the program name is MPDFnnn where:
M = MEDSTAT conversion program (constant)
P = Product
D = Data Sources
F = Function/Process
nnn = Sequence numbers (preferably in increments of 5)

Product:
D = DataScan

M = Performance Measurement Workstation (PMW)
P = Panorama
U = Utility

Data Sources:
C = Medical Claims (MCLM)
D = Drug Claims (DCLM)
E = Eligibility
M = Managed Care and Capitation
V = Provider

Function/Process:
0 = Pre-Convert
1 = Convert
2 = Install
3 = Edit
4 = Build
5 = Update
6 = Rolloff/Delete

Example:

MDD501 = First program used in the DataScan Drug Claims Update Process

The responsible Developer assigns the program number.

COMMENTS

Informational comments must be placed at the end of the Identification Division in a maintenance log. Following is an example of the maintenance log.

Modification Date	IR Number	Developer Name	Modification Description
97/09/15	999	Ron Howard	Modify drop conditions to keep and process eligibility records from Solano, San Mateo, and San Bernadino counties.
97/03/27	923	Suzy Smith	Added conversion map "AIDCODE" to be loaded to internal table; removed hard code.

Upon subsequent program revisions, the comments in the maintenance log must be revised and kept current. Additional comments (referencing the specific IR number) will be used throughout the program to clarify more complex routines.

PROGRAM DESCRIPTION

The program description must give a high level overview of the program's purpose (i.e., identify the main function of the program), and any special processing considerations.

Program revisions must be recorded in an orderly fashion and listed by date (most recent listed first) with a brief description of the purpose, and changes made. Include Investigation Request (IR) number, data of revision, and Developer Name.

1.8.1.2 Environment Division

INPUT-OUTPUT SECTION

DDNAMES must reflect file content to the extent possible with an 8 character limit. For example:

FILE	DDNAME
VITAL STATISTICS SUMMARY REPORT	SMRYRPT
VITAL STATISTICS BIRTH MASTER INPUT	MASTIN
VITAL STATISTICS BIRTH MASTER OUTPUT	MASTOUT

1.8.1.3 Data Division

GENERAL

The primary goal in organizing and coding data storage areas is to make it as easy as possible to locate data fields. The following rules apply to both the FILE SECTION and WORKING-STORAGE.

- Keep it readable:
  - ◊ Code one data name per line and indent each level of an '01' data description by 3-5 characters. Use blank lines to set off related groups of fields.
  - ◊ Start PICTURE values in the same column for ease of counting byte positions. For consistency, column 48 is recommended.
  - ◊ Ensure that PICTURE value formats are consistent for ease of counting byte positions.

For example:

```
01   XXXXX.
05  XXXXX                    PIC  X(023).
05  XXXXXXXX                 PIC  X(002).
05  XXXXXXX                  PIC  X(123).

01   XXXXXXXXXXX.
05  XXXXX.
10  XXXXX                    PIC  X(004).
10  XXX                      PIC  X(005).
```

Instead of:

```
01  XXXXX.
05  XXXXX  PIC  X(23).
05  XXXXXXXX  PIC  X(2).
05  XXXXXXX  PIC  X(123).
01  XXXXXXXXXXX.
05  XXXXX.
10  XXXXX  PIC  X(4).
10  XXX  PIC  X(5).
```

- Label modified code

If possible, identify code that was added or modified as a result of a program revision. Include the IR number so that it is possible to trace code back to the revision authorization.

WORKING-STORAGE SECTION

Organized and consistent presentation of working storage aids in researching and debugging production problems and assists in the modification of programs. It is recommended that this section be structured as follows:

- Have certain fields towards the top of WORKING-STORAGE to aid in reading dumps if a program terminates abnormally. For this reason the counters (input and output balancing counts followed by other counters), sequence or match keys, and tables appear at the beginning of WORKING-STORAGE.

- The second half of WORKING-STORAGE contains the record descriptions, detail lines and headers, miscellaneous fields, and abend-related fields. It is easier to modify and follow a program when data that are logically related or used for a similar purpose are grouped and presented in a familiar order. Thus, I/O fields or records are grouped together, print lines are grouped by report and are contiguous, etc.

Working-Storage Groups

- Use the literal constant ‘WORKING STORAGE STARTS HERE’ as the first statement in the WORKING-STORAGE section. The use of this literal will enable the trouble shooter to locate the beginning of the WORKING-STORAGE section in a core dump.

- Counters must follow the WORKING-STORAGE literal. Counters are defined as accumulators. The first counter(s) will be the total records read counter. This enables the trouble shooter to determine which record in the input file actually caused the problem. Other counters that may prove useful

to the programmer are records 'written' counter (this will help if the problem occurs in the output of the record), and records 'dropped' counter (if records are actually bypassed in the program).

- All working storage data elements must be organized so that like data elements are easier to find. When grouping "like" data elements, include a comment in the program identifying the type of data elements found in that group.

Field Formats

- Using PACKED-DECIMAL Fields

The machine stores PACKED-DECIMAL/COMP-3 fields in such a way that includes the sign in the lower order half byte. Therefore, a packed field that is defined with 6 digits and a packed field that is defined with 7 digits will both use 4 bytes in the machine. Inefficiencies may be introduced when an even number of digits are defined. That is why an odd number of bytes is highly recommended. A couple of examples follow:

```
INPUT-RECORD-READ      PICS  9(07)          COMP-3
DOLLAR-AMOUNT-PAID     PICS  9(09)V99PACKED-DECIMAL.
```

ITSD recommends the use of PACKED-DECIMAL rather than COMP-3 in coding. Although they provide the same function, the PACKED-DECIMAL is more descriptive.

- BINARY Fields

COMPUTATIONAL or BINARY fields are allocated by the system as either 2-byte or 4-byte fields, depending upon the number of digits in the PICTURE clause. Defining 1-4 digits results in a 2-byte field. Defining 5-9 digits results in a 4-byte field. In order to avoid possible truncation of significant digits, always define binary fields with either 4 or 9 digits. (It is possible to define a binary field that is 8 bytes long. However, the CPU cannot directly work with such a field and great inefficiencies are introduced).

ITSD recommends the use of BINARY rather than COMP in coding. Although they provide the same function, BINARY is more descriptive.

- Synchronized Fields

Use of synchronized fields (SYNC) is highly discouraged. SYNC causes the data element to start on a half-word or full-word boundary. This used to be required for binary fields, but is no longer needed.

Other Working-Storage Considerations

- Create group level (01 level) data names that will identify the function of data contained in the subsequent statements. For example:

```
01  COUNTERS.
    05   CLIENT-REC-READ-CNT
    05   CLIENT-MST-REC-WRITTEN-CNT
    05   GOOD-EDIT-REC-CNT
    05   CLIENT-REC-BYPASSED-CNT

01  SWITCHES.
    05   TRANS-EOF-SW
    05   MAST-EOF-SW

01  ERROR-DETAIL-LINE.
    05   FILLER                    PIC  X(002)
                                     VALUE SPACES.
    05   ERROR-TRANS-CODE PIC  X(001).
    05   ERROR-BATCH-NUM           PIC  X(023).

01  ERROR-REPORT-HEADER.
    05   FILLER                    PIC  X(011)
                                     VALUE  ' RUN DATE: '
    05   HL-RUN-DATE               PIC  X(008).
    05   FILLER                    PIC  X(033)
                                     VALUE SPACES.
    05   FILLER                    PIC  X(029)  VALUE
         'DEPARTMENT OF HEALTH SERVICES'.
```

- Identify the logical levels of tables and their indices with comments. If applicable, give a general explanation of table load/unload logic, especially conversion maps.

1.8.1.4 Procedure Division

Program Architecture

Structured programming is usually defined as containing the following elements:

- Top down design
- Functional modules
- Use of basic program structures (hierarchy, sequence, alternation, and repetition)
- Minimal or no use of GO TO or other unconditional transfer statements.

Other features of a structured program include single entry and exit points for procedures, use of common routines to reduce redundancy, and statement indentation to improve readability.

Top down design is design that proceeds from the general to the specific in an ordered hierarchy. Major processes are identified and broken down into successively smaller functional units. The overall flow of the program must be discernible from the main logic or higher order procedures.

Functional modules are paragraphs that are usually devoted to a single function and have a limited purpose (e.g. to build an output record, search a table, etc.).

The basic programming structures have been defined as:

- Hierarchy — program code proceeds from the general to the specific according to function

- Sequence — statements are executed in a predetermined sequence (i.e., top to bottom). Sequence implements processing steps that are essential in the specification of algorithms.

- Alternation — (i.e., IF-THEN-ELSE, EVALUATE) conditional execution of statements, paragraphs or sections. Provides the facility for selected processing based on some logical occurrence.

- Repetition — conditional iteration of paragraphs or sections (e.g., PERFORM UNTIL statements). Manages controlled looping.

Do not use “GO TO” or other unconditional transfer statements. Unconditional transfer statements are not supported by structured programming techniques. If not used properly, they can cause multiple exits from paragraphs (since there is no provision for a return to the next statement after the “GO TO”). Also, “GO TO” logic is difficult to follow when performing program maintenance.

Avoid use of the PERFORM THRU construct for the same reasons as mentioned above.

Standard Paragraph Names

The standard paragraph name is constructed in the following format:

9999-PARAGRAPH-ID.

9999 = a four-digit number which is in ascending sequence from the beginning of the program. Where program complexity allows, number the paragraphs by 100's, to reserve the additional numbers for a new paragraph to be inserted in their logical place when the program is modified.

For example:

```
1000-INITIALIZATION.
1100-EDIT-PARAMETER.
1200-LOAD-COUNTY-TABLE.
1300-EMPTY-MASTER-CHECK.
```

PARAGRAPH-ID is constructed of a character string containing up to twenty-five (25) characters, including hyphens. The paragraph-ID identifies what major activity or function takes place in the particular paragraph. When numbering the paragraphs in a program, it is very helpful to keep all of the paragraphs that are related within a major numbering grouping. Utilizing the correct numbering format will enable a trouble-shooter to follow the flow of data through the program.

An example of the numbering scheme follows:

```
PROCEDURE DIVISION

0000-MAIN-DRIVER.
    PERFORM 1000-INITIALIZE.
    PERFORM 4000-PROCESS-DATA UNTIL END-OF-FILE.
    PERFORM 9000-DISPLAY-CONTROL-TOTALS.
    GOBACK.

1000-INITIALIZE.
    DISPLAY  '**********  MDPU500   **********'.
    DISPLAY SPACE.
    OPEN   INPUT  INPUT-FILE
           OUTPUT  OUTPUT-FILE.
    PERFORM 2000-READ-INPUT-FILE.

2000-READ-INPUT-FILE.
    READ  INPUT-FILE
        INTO INPUT-RECORD
        AT END
            SET  INPUT-EOF TO TRUE
    END-READ.
    IF  INPUT- EOF
        CONTINUE                    (preferred usage is
                                     CONTINUE',
                                not 'NEXT SEQUENCE')
    ELSE
        ADD +1 RECORDS-READ-CNT
    END-IF.
```

```
4000-PROCESS-DATA.
    PERFORM 4100-PROCESS-RECORD.
    PERFORM 2000-READ-INPUT-FILE.

4100-PROCESS-RECORD.
    MOVE INPUT-RECORD TO OUTPUT-RECORD.
    PERFORM 4200-WRITE-OUTPUT-RECORD.

4200-WRITE-OUTPUT-RECORD.
    WRITE OUTPUT-RECORD.
    ADD +1 TO RECORDS-WRITTEN-CNT.

9000-DISPLAY-CONTROL-TOTALS.
    DISPLAY 'RECORDS READ   = ' RECORDS-READ-CNT.
    DISPLAY 'RECORDS WRITTEN  = ' RECORDS-WRITTEN-CNT.
    CLOSE  INPUT-FILE
           OUTPUT-FILE.
```

1.8.1.5 Additional Comments

Use of Case Statements

Use “EVALUATE” rather than “IF-THEN-ELSE” constructs to manage complex logic.

Moderate Paragraph Lengths

Maintain a single function per paragraph. Long paragraphs usually perform more than one logical function. Shorter, logical functions improve future understanding, maintenance and enhancement of programs. The general rule is to avoid paragraphs longer than one page. When long paragraphs are unavoidable, ensure adequate comments to assist in simplification of the code.

Readability

Code one verb or phrase per line. Indent each sub-phrase at a lower hierarchy by three to four characters.

For example:

```
READ TRANS-FILE INTO TRANS-RECORD
     AT END SET TRANS-EOF TO TRUE
END-READ.
```

Instead of:

```
READ TRANS-FILE INTO TRANS-RECORD
AT END SET TRANS-EOF TO TRUE.
```

- Make use of the verb delimiters (end-if, end-perform, etc.) especially on verbs that can use other verbs in sub-phrases such as IF, EVALUATE and READ. This also helps to avoid logic errors caused by missing periods.

- Group like code when appropriate. For example, in a print routine, group the WRITE statements using blank lines to separate the WRITE statements from the print heading check routine.

- In a paragraph with a loop process, if there is not a separate BEGIN, BODY, and END paragraph, identify with comments each separate process within the one paragraph. This condition exists when there are not enough statements to separate the BEGIN and END process into paragraphs and the BODY may be an IN-LINE PERFORM.

Use Of Relational Operators

- Promote use of positive logic flow. Avoid negative logic using the word "NOT", unless it is simple and easily understood. If there is a clear positive method of expressing the condition, the positive condition is always preferred. Negative logic is sometimes difficult to understand and difficult to debug. This is especially true in complex situations.

- Complex compound conditions are also discouraged if there is a reasonable alternative.

Example #1:

Poor Coding:
```
IF A NOT <  B
   PERFORM 1234-PROCESS
END-IF.
```

Preferred:
```
IF  A >=  B
   PERFORM 1234-PROCESS
END-IF.
```

Example #2:

Poor Coding:
```
IF A = B OR A NOT <  C AND C =  D
   PERFORM 1234-PROCESS
END-IF.
```

Preferred:
```
IF A = B
      PERFORM 1234-PROCESS
   ELSE
      IF A  >= C  and C = D
         PERFORM 1234-PROCESS

                              END-IF
   END-IF
```

Common Coding Issues

- Set or increment counters after the event has occurred for consistency. If an abend results in a core dump, it is easier to interpret the meaning of the counters if all Developers use a standard position for incrementing the counts.
- In editing situations, move data after it has been edited within the same module, unless it is part of a separate move paragraph.

Program Accountability

- Use DDname "CNTLRPT" for all balancing and control reporting. Avoid use of "DISPLAY" in production programming.
- Print meaningful end-of-job counts at the end of processing. Record counts will be reported for all input and output files (not necessarily reports) if they are processed sequentially.

- Print a complete accounting of update activities (transactions rejected, records added, records changed, records deleted, etc.), if one or more files are updated, so that all output can be reconciled to the input counts.
- Print balancing accumulators for NORMAL program termination. Code all balancing instructions that provide indicators to evaluate problematic processing, plus a message indicating that the counts balance or do not balance.
- Report meaningful abnormal termination messages before the control totals. The message must be descriptive and provide enough information to assist in problem resolution.
- If a program ends by detecting an abnormal or error condition, the program must print all termination messages, all control totals, establish appropriate condition return code, then call "CSBERROR" (see Conversion Programming Guide).

For COBOL programs, use return codes to indicate severity level of problems encountered:

0 = No errors; continue processing
4 = Informational
8 = Warning
12 = Severe
16 = Critical

These codes may be used to handle conditional branching in job execution.

Recommended Usage of Sub-scripts and Indices

Sub-scripting and indexing are excellent tools for table handling, and must be used thoughtfully. Consider the following points:

- Indexing is more efficient than sub-scripting for table searches or when the same index is used several times relative to the number of times its value changes.
- Use an index to reference only the table in which it is defined. In terms of sub-scripting, do not use the same subscript for referencing different tables or data names unless the subscript is used to access both fields or tables in the same process with the same value. Avoid using the same subscript in unrelated parts of the program.
- Sub-scripting is useful when an existing data field contains the exact value needed to identify the correct location in the table. For example, if county 19 is always in the 19th table location you can use the county code from an input record as the subscript. It is also easier to use subscripts when doing coordinated processing of two or more tables and the same subscript values are used to access all of the tables.
- For files with detail segments or for table items referenced several times in the same logic loop, it is better to move the segment or table item desired into a WORKING-STORAGE area and to reference the individual data elements

from there, rather than using subscripts or indexes to access each data field individually in the table. This method is more efficient and easier to follow.

- Give the subscript/index meaningful data names, such as 'AID-CODE-SUB' or 'COUNTY-TABLE-IDX'.
- Specify 'BINARY' and 'SIGNED' in the picture clause of subscripts. Conversion to BINARY will occur when BINARY is not specified. Designating the BINARY field as 'SIGNED' saves an extra instruction. For unsigned BINARY fields, an extra instruction is required to remove the sign whenever the value is changed.

1.9 Exit Criteria

Agreement by all members of the Core Team effected by the policy/process is required before the policy/process is implemented. Each Core Team member must be prepared to sign-off on the policy/process and inform their team members that they will support the implementation and enforcement of the content.

1.9.1 Exit Exception Criteria

If an applicable program cannot comply with a Standard in this section, the issue must be documented in the program code, and in the program design documentation. These items are presented at code walk-through where alternatives may be presented. All remaining exceptions must be approved by the Project Development Manager.

1.9.2 Exit Exception Handling

The exception must be documented and agreed to by the Project Development Manager.

1.10 Forms and Subject Examples

N/A

1.11 Reference Material

The following materials may be used for reference purposes to add clarity to any Standard or process outlined in this document, or as required by the Developer.

COBOL II Reference Manual
JCL Reference Manaual
SyncSort Reference Manual
JCL Standards
MEDSTAT Conversion Programming Manual

1.12 Policy History

Established/Revision Date	Established/Revised By	Change Description
4/3/2000	John Mulcahy	Revised to new format, added clarifying information to some sections
8/1/98	Barbera Bridgewater	Policy/Process Established